**ASPECTOS AVANZADOS DE OOP**

**Encapsulamiento**

**Definición**

El encapsulamiento habla del modo de ocultar como ha sido implementado el estado, los atributos, de un objeto. Se accede a este estado a través de los métodos públicos, es decir su interfaz pública. Una buena práctica es hacer las validaciones correspondientes a los posibles estados del objeto, en estos métodos, de modo tal de mantener al objeto en un estado consistente.

También se lo llama "information hiding". De la misma forma podemos respetar el encapsulamiento si se tiene, en la clase Auto, un atributo privado velocidad, que sea privado, luego el único modo de modificar la velocidad es a través del método acelerar() y frenar(), es decir que está encapsulada la velocidad, y solo se la modifica por los métodos acelerar() y frenar(), no se puede cambiar la velocidad de ningun otra forma.

Ejemplo en Java:

public class Persona{

    // Atributos

    private int altura;

    // Metodos

    public int getAltura(){

          return this.altura;

        }

        public void setAltura(int unaAltura){

          this.altura = unaAltura;

    }

  }

**Métodos de acceso**

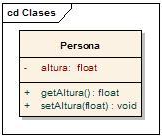
Los métodos de acceso son el medio de acceder a los atributos privados del objeto. Son métodos públicos del objeto.

**El "getter"**

El método para acceder a los atributos en forma de solo lectura se los denomina "getters". Son los métodos que retornan el valor de los atributos. El NetBeans, como la mayoría de los entornos de desarrollo, permite generarlos de forma automática.

**El "setter"**

El método para acceder a los atributos en forma de escritura se los denomina "setters". Son los métodos que establecen el valor de los atributos. También se los genera de forma automática en los entornos de desarrollo.



**Constructores y Destructores**

**El constructor**

Los constructores son métodos pertenecientes a la clase. Se utilizan para construir o instanciar una clase. Pueden haber varios constructores, de acuerdo a las necesidades del usuario.

Ejemplo en Java:

  public class Persona{

    //Atributos

    private int altura;

    // Constructores

    public Persona{

    }

    public Persona(int unaAltura){

      this.altura = unaAltura;

    }

  }

**El destructor**

El destructor se utiliza para destruir una instancia de una clase y liberar memoria. En Java no hay destructores, ya que la liberación de memoria es llevada a cabo por el Garbage Collector cuando las instancias de los objetos quedan desreferenciadas. El método dispose() de cada objeto se llama previo a ser "recolectado".

**Atributos y Métodos Estáticos**

**Atributos de Clase**

Un atributo de clase es un atributo común a todos los objetos instanciados de la clase. Al estar definido en la clase no hace falta instanciar la clase para utilizarlo. Las constantes se suelen definir como atributos de clase.

Ejemplo:

El atributo cantidadDeOjos en la clase Persona, debido a que todas las instancias de la clase persona tendrán igual cantidad de ojos.

Ejemplo en Java:

  public class Persona{

          public static int cantidadDeOjos = 2;

        }

Por lo tanto, en su uso:

Persona mario = new Persona();

  Persona maria = new Persona();

  mario.cantidadDeOjos // es igual a 2

  maria.cantidadDeOjos // es igual a 2

  Persona.cantidadDeOjos // es igual a 2

y este 2 es siempre el mismo, no se instancia un cantidadDeOjos para cada objeto, siempre es el mismo 2 de cantidadDeOjos.

Si modifico esta cantidad de ojos, se modifica para todas las instancias del objeto:

  Persona.cantidadDeOjos = 3;

entonces

  mario.cantidadDeOjos // es igual a 3

  maria.cantidadDeOjos // es igual a 3

**Métodos de Clase**

Los métodos de clase son un comportamiento común a todas las instancias que pertenecen a la misma clase. Al ser un método de clase, no hace falta instanciar un objeto de la clase para utilizarlo. Estos métodos no hablan del estado actual de la clase, sino solo de un comportamiento genérico de la clase, de un procedimiento que solo utiliza los parámetros de entrada o las variables estáticas. Son para un uso particular y es bien visible cuando un método debe ser estático.

Ejemplo en Java:

 public class Persona{

        public static int obtenerAlturaMaxima(){

          // Esto es un comentario. Aqui va la definicion del metodo.

    }

  }

**Herencia**

**Definición**

La herencia se produce a partir de dos clases relacionadas, es decir una subclase que hereda los atributos y los métodos de la superclase. La jerarquía de clases, o árbol de herencia se lleva a cabo a partir de esta herencia entre objetos. Se puede entender como la relación "es un" entre dos clases. Si la subclase C2 hereda de la superclase C1, se dice que C2 es un C1.

Ejemplo:

La clase Hombre hereda de la clase Persona, entonces podemos decir que Hombre "es una" Persona.

Ejemplo en Java:

 public class Hombre extends Persona{

  }

Todas las clases en Java heredan de la clase Object.

**Qué es el Casting**

El casting es un procedimiento para transformar una variable primitiva de un tipo a otro, o transformar un objeto de una clase a otra clase siempre y cuando haya una relación de herencia entre ambas.

Existen distintos tipos de castings de acuerdo a si se utilizan tipos de datos o clases.

**Casteo Implicito (Widening Casting)**

El Casteo Implícito radica en que no se necesita escribir código para que se lleve a cabo. Ocurre cuando se realiza una conversión ancha - widening casting - es decir, cuando se coloca un valor pequeño en un contenedor grande.

Por ejemplo:

// Define una variable del tipo int con el valor 100

int numero = 100;

// Define una variable del tipo long a partir de un int

long numero2 = numero;

**Casteo Explicito (Narrowing Casting)**

El Casteo Explícito se produce cuando se realiza una conversión estrecha - narrowing casting - es decir, cuando se coloca un valor grande en un contenedor pequeño. Son susceptibles de pérdida de datos y deben realizarse a través de código fuente, de forma explícita.

Por ejemplo:

// Define una variable del tipo int con el valor 250

int numero = 250;

// Define una variable del tipo short y castea la variable numero

short s = (short) numero;

**Upcasting**

El upcasting se produce a nivel objetos. Suponiendo que existe una clase Empleado y clase Ejecutivo, la cual es una subclase de ésta. Un Ejecutivo entonces ES UN Empleado, y se puede escribir de la siguiente forma:

// Instancia un ejecutivo en una variable de tipo Empleado

Empleado e1 = new Ejecutivo("Máximo Dueño" , 2000);

A esta operación en donde un objeto de una clase derivada se asigna a una referencia cuyo tipo es alguna de las superclases se la denomina upcasting.

Cuando se realiza este tipo de operaciones, hay que tener cuidado porque para la referencia e1 no existen los atributos y métodos que se definieron en la clase Ejecutivo, aunque la referencia apunte a un objeto de este tipo.

Por el contrario, si ahora existe una variable de tipo Empleado y se desea acceder a los métodos de la clase derivada - suponiendo que contiene un método ejecutarPlanificacion() en la clase Ejecutivo - teniendo una referencia de una clase base, como en el ejemplo anterior, hay que convertir explícitamente la referencia de un tipo a otro. Esto se hace con el operador de cast de la siguiente forma:

// Instancia un ejecutivo en una variable de tipo Empleado

Empleado emp = new Ejecutivo("Máximo Dueño" , 2000);

// Se convierte (o castea) la referencia de tipo

Ejecutivo ej = (Ejecutivo) emp;

// Se aumenta el sueldo del ejecutivo

ej.ejecutarPlanificacion();

La expresión de la segunda línea convierte la referencia de tipo Empleado asignándola a una referencia de tipo Ejecutivo. Para el compilador es correcto porque Ejecutivo es una clase derivada de Empleado. En tiempo de ejecución la JVM convertirá la referencia si efectivamente emp apunta a un objeto de la clase Ejecutivo, caso contrario lanzará una excepción.

Por el contrario, si se intenta realizar lo siguiente:

  Empleado emp = new Empleado("Javier Todudas" , 2000);

   Ejecutivo ej = (Ejecutivo)emp;

No dará problemas de compilación, pero al ejecutar se producirá un error ya que la referencia emp apunta a un objeto de clase Empleado y no a un objeto de clase Ejecutivo.